고성능 병렬 CRC 생성기 설계

정회원 이현빈*, 박성주*, 민병우*, 박창원**

A Design of High Performance Parallel CRC Generator

Hyunbean Yi*, Sungju Park*, Pyoungwoo Min*, Changwon Park

요 약

본 논문은 통신 시스템에서 오류 검출을 위해 널리 사용되고 있는 Cyclic Redundancy Check (CRC) 회로의 병렬 구현을 위한 새로운 회로 축소 알고리즘 및 설계 기술을 소개한다. 논리 수준을 최소화하여 CRC 속도를 증진시키기 위해서 입력데이터와 CRC 내부 신호를 두 개 단위로 그룹화 하는 새로운 알고리즘을 개발하였다. 성능 평가를 위해 16비트와 32비트 CRC를 PLD (Programmable Logic Device) 및 표준 셀 라이브러리를 이용하여 합성하였으며, 기존에 제시되었던 방법보다 성능이 향상되었음을 보여준다.

Key Words : CRC(Cyclic Redundancy Check), Parallel CRC, Logic Optimization, XOR Gate

ABSTRACT

This paper presents an optimization algorithm and technique for designing parallel Cyclic Redundancy Check (CRC) circuit, which is most widely adopted for error detection. A new heuristic algorithm is developed to find as many shared terms as possible, thus eventually to minimize the number and level of the exclusive-or logic blocks in parallel CRC circuits. 16-bit and 32-bit CRC generators are designed with different types of Programmable Logic Devices, and it has been found that our new algorithm and architecture significantly reduce the delay.

Ⅰ. 서 론

Cyclic Redundancy Check (CRC)는 통신 시스템에서 가장 널리 사용되는 수신 데이터 오류 검출 방법이다. 여러 시스템에서, 한 비트 오류 검출 방법인 패리티 (Parity)나 한 바이트에 3-4개의 비트를 사용하여 오류를 검출하는 해밍코드 (Hamming Code)가 많이 사용 되어왔다[1]. 그러나, 고속의 중장거리 데이터 송수신시에 발생하는 대부분의 에러는 연속적인 여러 비트에 걸쳐 발생하는 특성이 있으며, 커다란 메시지 또는 패킷 단위의 전송시에 해밍코드를 사용하면, 오류 검출을 위한 오버헤드가 너무 커진다. 따라서, 데이터의 크기에 상관없이 코드생성 다항식 (Polynomial)의 차수만큼의 비트열로 여러 비트의 오류를 검출 할 수 있는 CRC는 고속 통신 시스템에 매우 적합하다[1,2]. 이러한 통신 시스템 기술의 발달로 네트워크 및 데이터I/O 송수신 표준들은1 Gb/s를 넘어 10 Gb/s 이상의 속도를 목표로 하고 있으며, 그에 따라 고속으로 동작 할 수 있는 CRC 회로 설계 기술이 필요하다.

CRC는 기본적으로, Linear Feedback Shift Register (LFSR)을 이용한 직렬 회로로 구성되어, CRC 코드를 생성하기 위해 직렬 데이터 입출력 속도에 맞는 비트 클럭을 사용해야 하거나, CRC 코

* 한양대학교 컴퓨터공학과 멀티미디어 시스템 연구실(bean@hanyang..ac.kr),
* 한양대학교 컴퓨터공학과 멀티미디어 시스템 연구실(packsj@hanyang..ac.kr),
* 한양대학교 컴퓨터공학과 멀티미디어 시스템 연구실(minpw@hanyang..ac.kr),
** 전자부품연구원 유비쿼터스컴퓨팅 연구센터(parkcw@keti.re.kr)
논문번호 : 040154-0423, 접수일자 : 2004년 월 일
※본 논문은 유망전자부품기술개발사업 채널 기반형 입출력 부품개발 과제로부터 지원을 받아 진행하였습니다.

드 생성 지연시간을 위한 버퍼링이 필요하다. 따라서, 고속으로 갈수록 구현이 어려워지므로, 병렬 CRC 구현에 관한 많은 연구가 이루어지고 있다 [3,4,5,6,7,8]. 병렬 CRC 구현을 위해서는 표 검색 (table look-up) 방식과 Exclusive-OR (XOR) 조합 회로를 고려 할 수 있는데, 표 검색 방식은 코드 생성 다항식의 차수가 하나씩 증가함에 따라서 테이블의 크기가 두 배씩 커져야 하므로, 확장성과 표 검색을 위한 지연시간 증가에 문제가 있다.

본 논문에서는 XOR 조합으로 이루어진 병렬 CRC 회로에서 가능한 한 많은 XOR 게이트를 공유하도록 하여 게이트의 수를 줄이기 위한 휴리스틱 알고리즘과, 임계 경로의 논리 레벨을 최소화 할 수 있는 설계 구조를 제시하고, 하드웨어로 설계하여 성능 평가를 한다. 2장에서 기본적인 병렬 CRC 구현방법 및 기존연구에 대해서 살펴보고, 3장에서 본 논문에서 제시하는 알고리즘을 설명한다. 4장에서 논리 레벨을 고려한 하드웨어 구조를 제시하며, 5장에서는 기존의 다른 알고리즘에 의한 결과와 성능을 비교하고, 6장에서 결론을 맺는다.

Ⅱ. 기존의 병렬 CRC 알고리즘

CRC는 데이터 송수신 시스템에서 가장 많이 사용되는 오류 검출 방법이다. 두 시스템 간에 미리 약속된 특정수를 제수로 사용하여, 송신부에서 보내려는 데이터를 그 수로 나누고, 나눈 나머지를 데이터의 끝에 함께 실어 전송하면, 수신 측에서는 받은 데이터를 같은 수로 나누어 나머지를 비교하거나 나머지까지 포함한 데이터 전체를 나누어 나머지가 '0'이 되는지를 파악하여 데이터의 오류 유무를 판단한다. 여기서, 데이터를 나눈 나머지 자체가 CRC 코드가 되며, 어떤 특정 제수에 의한 나머지 값이므로 CRC 코드는 데이터 크기에 상관없이 항상 그 제수의 이하가 된다[2].

이러한 개념을 바탕으로, 현재 여러 통신 시스템에서는, 제수로써 표준화 되어있는 몇 가지 다항식 (polynomial)을 주로 사용하며, 이진 모듈로 2 연산을 수행함으로써 나온 결과를 CRC 코드로 사용한다. 따라서, 직렬 데이터 송수신 시스템에서, XOR 게이트를 이용하여 이진 모듈로 2 연산을 수행 할 수 있으므로, <그림 1>과 같이 LFSR과 XOR 게이트를 이용하여 하드웨어로 구현 할 수 있다.

이와 같은 직렬 CRC 생성은 시스템 속도 향상에 매우 큰 걸림돌이 된다. 따라서, 직렬 데이터를 병렬로 변환 후, 한 클럭에 CRC 코드를 생성하려는 연구가 요구되었으며, 그 결과, 여러 번의 쉬프트와 XOR 연산 후에 각 플립플롭에 저장되었던 CRC 코드를, 병렬 데이터 입력을 XOR 조합 회로를 통해 한 사이클에 생성될 수 있는 회로를 개발하게 되었다. 기본적인 병렬 CRC 회로 설계 방법은 [1]에 잘 설명 되어 있다.

병렬 데이터의 크기와 CRC 레지스터의 길이가 모두 양의 정수 n인, n-비트 CRC 생성기 구현을 위한 알고리즘을 설명하기 위하여, n 비트 LFSR을 놓고 아래와 같이 몇 가지 기호를 정의하자.

i : 1 이상 n 이하인 양의 정수
Fi : LFSR의 i 번째 레지스터
Ci : 쉬프트를 시작하기 이전에 Fi 에 저장되어 있는 초기값
Di : i 번째 입력 데이터

데이터는 최하위 비트인 D1부터 입력되고, 쉬프트는 그림 1 과 같이 번호가 낮은 레지스터 방향으로 진행된다고 하자. 그러면, 쉬프트를 수행하기 전의 레지스터의 초기값은 Ci 가 되고, 매 클럭마다 한 비트씩 쉬프트를 하여, Fi 에는 쉬프트와 XOR 게이트에 의해 C 와 D 의 조합으로 이루어진 새로운 값이 저장된다. 예를 들어, 그림 1 에서한번의 쉬프트 후, F16은 (D1 xor C1), F11은 (D1xor C1 xor C12), F4는 (D1 xor C1 xor C5), 그 밖에 다른 레지스터는 단순 Fi+1의 값을 저장하게 된다. 이와 같이 하여, n 번 쉬프트 후에 레지스터에 저장되는 값이 n 비트 데이터에 대한 CRC 코드가 된다. 게다가, Xi를 (Di xor Ci)라고 하면, n 번 쉬프트 후,

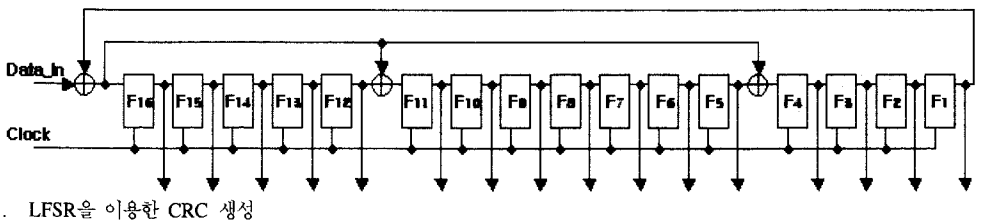


그림 1. LFSR을 이용한 CRC 생성

레지스터의 결과 값이 모두 Xi만의 조합으로 이루어진다는 특징이 있다. 따라서 Xi하나가 2입력 XOR 게이트가 되고 Xi의 조합으로부터 한 클럭에 CRC 코드를 생성하는 병렬 CRC 회로를 구성할 수 있다. 그림 2 는 n번의 쉬프트 후 각 레지스터의 결과를 생성하는 Xi의 조합을 이끌어 내기위한 알고리즘의 유사코드이다. 여기에 매번 루프를 실행할 때마다 각 F에 저장되는 결과를, 계산된 숫자가 아닌 Xi로 저장하여 나타냄으로써 원하는 Xi의 조합을 구할 수 있다.

```
Definitions
n : Size of data and register
D : Input data
C : Initial value of register
X : D xor C
P : n bit parameter
    if xor tap is needed, then '1'
    if xor tap is not needed, then '0'
    (it depends on the polynomial used)
F : Result value generated
CRC Gen :
for (i=1;i=n;i++) {
   X[i] = D[i] ^ C[i];
   for(j=1; j=n; j++){
          if(P[j] == 1)
            F[j] = X[i] ^ F[j+1];
          else
                F[j] = F[j+1];
   }
}
```

그림 2. CRC 생성 알고리즘

그림 1 의 16비트 CRC에 그림 2의 알고리즘을 적용시킨 후의 F에 대한 X의 조합은 표 1과 같다. X는 데이터 D와 초기값 C의XOR을 나타낸다. 따라서, F6의 경우 8개의 XOR 조합으로 이루어짐을 알 수 있다.

논문 [3]에서 Tong-Bi Pei 와 Charles Zukowski 는 8번 쉬프트 후의 결과를 이용하여 8비트 병렬 데이터를 입력으로 하는, 32 비트 CRC 회로를 설계했으며, Richard F. Hobson과 Keith L. Cheng은 위 논문을 바탕으로 레이아웃 라우팅의 복잡도를 줄이기 위하여, 프리 디코딩 로직과 이진 트리 축소 기법을 사용하여 속도를 향상시켰다[4]. M. S. Joshi

표 1. Cumulated X[i] for Fi after 16 shif

F1	X4 X5 X8 X12 X16
F2	X3 X4 X7 X11 X15
F3	X2 X3 X6 X10 X14
F4	X1 X2 X5 X9 X13
F5	X1 X4 X8 X12
F6	X3 X4 X5 X7 X8 X11 X12 X16
F7	X2 X3 X4 X6 X7 X10 X11 X15
F8	X1 X2 X3 X5 X6 X9 X10 X14
F9	X1 X2 X4 X5 X8 X9 X13
F10	X1 X3 X4 X7 X8 X12
F11	X2 X3 X6 X7 X11
F12	X1 X2 X5 X6 X10
F13	X1 X8 X9 X12 X16
F14	X7 X8 X11 X15
F15	X6 X7 X10 X14
F16	X5 X6 X9 X13

등은 표 검색 방법을 이용한 알고리즘을 제시하였다[5]. 논문[6]은 Galois field 이론을 바탕으로 병렬 CRC 회로를 하드웨어로 구현하였으며, Giuseppe Campobello 등이 논문 [8]에서, 논문 [6]을 개선하고 일반화 하여 기존의 논문에서 제시한 설계 결과에 비해 좋은 성능을 보임을 입증 하였으며, 특히 고속으로 CRC 코드를 생성하기 위하여 변형된 LFSR을 제시한 논문[7]과의 구체적인 비교를 제시하였다.

Ⅲ. 휴리스틱 CRC XOR 회로 축소 알고리즘

2장의 결과를 그대로 회로로 구현 할 수 있으나, 최적화를 거치지 않았기 때문에 좋은 성능을 기대하기 어렵다. 게다가, 최적화 알고리즘 또한 NP-Hard 문제로써 회로의 특성에 따라 경험적인 방법을 적용할 필요가 있다. 이 장에서, F와 X의 관계를 바탕으로 하여, 임계 경로를 최소화 하면서 최대한 많은 게이트를 공유하도록 하는 알고리즘을 제시한다.

단지 논리 레벨만을 줄이고자 한다면, 다중 입력 게이트를 사용함으로써 얼마든지 가능하다. 그러나, FPGA 구현 및 일반적인 표준 셀 라이브러리의 논리 게이트의 입력 수가 제한적이라는 점을 고려하여, 알고리즘을 통해 생성되는 XOR 게이트의 최대 입력 수는 2로 제한한다. 먼저 아래와 같이, 1 이상 n 이하인 정수 i 와 j 에 대하여, 행은 Fi, 열은 Xj 를 의미하는 n x n 행렬 G를 정의한다.

$$G = \begin{pmatrix} g_{11} & g_{12} & g_{13} & \cdots & g_{1n} \\ g_{21} & g_{22} & g_{23} & \cdots & g_{2n} \\ g_{31} & g_{32} & g_{33} & \cdots & g_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & g_{n3} & \cdots & g_{nn} \end{pmatrix} \quad g_{ij} \begin{cases} \text{'1', if } F_i \text{ has } X_j \\ \text{'0',} \end{cases} \quad (1)$$

예를 들어, 표 1은 (2)와 같은 행렬로 나타낼 수 있다.

$$G = \begin{pmatrix} 0&0&0&1&1&0&0&1&0&0&0&1&0&0&0&1 \\ 0&0&1&1&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&1&1&0&0&1&0&0&0&1&0&0&0&1&0&0 \\ 1&1&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\ 1&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0 \\ 0&0&1&1&1&0&1&1&0&0&1&1&0&0&0&1 \\ 0&1&1&1&0&1&1&0&0&1&1&0&0&0&1&0 \\ 1&1&1&0&1&1&0&0&1&1&0&0&0&1&0&0 \\ 1&1&0&1&1&0&0&1&1&0&0&0&1&0&0&0 \\ 1&0&1&1&0&0&1&1&0&0&0&1&0&0&0&0 \\ 0&1&1&0&0&1&1&0&0&0&1&0&0&0&0&0 \\ 1&1&0&0&1&1&0&0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&1&1&0&0&0&1&0&0&1 \\ 0&0&0&0&0&0&1&1&0&0&1&0&0&0&1&0 \\ 0&0&0&0&0&1&1&0&0&1&0&0&0&1&0&0 \\ 0&0&0&0&1&1&0&0&1&0&0&0&1&0&0&0 \end{pmatrix} \quad (2)$$

이 행렬의 원소 gij는, Fi가 Xj를 포함하고 있으면 '1' 이고, 그렇지 않으면 '0' 이다. 이 행렬G를 바탕으로 다음과 같은 순서로 알고리즘을 수행한다.

1 단계: '1'의 개수가 가장 많은 열을 찾고, 그 열을 A라고 하자.

2 단계: A열을 나머지 다른 모든 열과 비교하되, 그 중, A와 공통으로 '1'을 포함하고 있는 같은 행의 수가 가장 많은 열을 찾고, 그 열을 B라고 하자.

3 단계: A열과 B열에서, 같은 행에 공통으로 포함되어 있는 '1'을, 두 열에서 모두 삭제 한다. 삭제 후 '1'의 개수가 줄어든 새로운 열을 각각 A*와 B*라고 하자.

4 단계: 행렬 G에서 A열과 B열을 각각 A*열과 B*열로 교체하고, 열이 교체된 새로운 행렬을 G*라고 하자.

5 단계: 행렬의 각 행에 '1'이 하나 이하가 될 때까지 1"4 단계를 반복한다.

위의 과정 중 몇 가지 선택의 문제가 발생할 수 있다. 1 단계에서 찾은 열이 두개 이상인 경우에, 그 중 j가 낮은 열을 우선으로 결정하도록 하였다. 2 단계에서도 마찬가지로, B열로 선택될 수 있는 후보 열이 두 개 이상 존재할 수 있다. 이 경우에는, 일단 후보열 중 '1'의 개수가 가장 많은 열을 택하되, 그러한 열 또한 두개 이상 존재하면, 그 중 j가 가장 낮은 열을 선택하도록 하였다. 어느 열을 선택하느냐에 따라 결과가 달라질 수 있으며, 이러한 경우의 선택 방법에 관한 문제는 논외로 한다.

이해를 돕기 위하여, (2)의 행렬을 바탕으로 이 논문에서 제시한 알고리즘을 적용해 보자.

1 단계: 1st, 2nd, 3rd, 4th, 5th, 6th, 7th, 8th 열이 각각 7개씩 가장 많은 '1'을 포함하고 있다. 따라서, j가 가장 낮은 1st 열을 A열로 놓는다.

2 단계: 2nd, 5th, 9th 열이 A열과 각각 4행에 공통으로 가장 많은 '1'을 포함하고 있다. 따라서, 그 중 j가 가장 낮은 2nd 열을 B열로 놓는다.

3 단계: A열과 B열에서 같은 행에 있는 4개의 '1'을 '0'으로 치환하고, 바뀐 두 열을 각각 A*열과 B*열로 놓는다. 여기에서, A열과 B열에서 같은 행의 '1'의 삭제가 하나의 2 입력 XOR 게이트 생성을 의미한다.

4 단계: (2)의 행렬 G에서 A열과 B열을 각각 A* 열과 B*열로 교체하고, 열이 교체된 새로운 행렬을 (3)과 같이 G*로 놓는다.

$$G^* = \begin{pmatrix} 0&0&0&1&1&0&0&1&0&0&0&1&0&0&0&1 \\ 0&0&1&1&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&1&1&0&0&1&0&0&0&1&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0 \\ 1&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0 \\ 0&0&1&1&1&0&1&1&0&0&1&1&0&0&0&1 \\ 0&1&1&1&0&1&1&0&0&1&1&0&0&0&1&0 \\ 0&0&1&0&1&1&0&0&1&1&0&0&0&1&0&0 \\ 0&0&0&1&1&0&0&1&1&0&0&0&1&0&0&0 \\ 1&0&1&1&0&0&1&1&0&0&0&1&0&0&0&0 \\ 0&1&1&0&0&1&1&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&1&0&0&1&0&0&0&1&0&0&0 \\ 1&0&0&0&0&0&0&1&1&0&0&0&1&0&0&1 \\ 0&0&0&0&0&0&1&1&0&0&1&0&0&0&1&0 \\ 0&0&0&0&0&1&1&0&0&1&0&0&0&1&0&0 \\ 0&0&0&0&1&1&0&0&1&0&0&0&1&0&0&0 \end{pmatrix} \quad (3)$$

5 단계: (3)의 행렬 G*을 각 행에 '1'이 하나 이하가 될 때까지 1"4 단계를 반복한다.
최종적으로 (4)의 행렬과 같은 결과를 얻을 수 있으며, 삭제되지 못하고 남은 '1'은 팬-아웃을 의미한다.

$$G = \begin{pmatrix} \cdots & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 1 & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 1 & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} \quad (4)$$

Ⅳ. 논리 레벨을 고려한 H/W CRC 생성기 구조

완전 주문형 (Full-Custom) 방식으로 설계하지 않는 한, 게이트의 선택 및 논리 레벨은 합성 툴 및 라이브러리에 의존하여야 한다. 따라서, 논리 레벨을 줄이기 위하여 그림 3과 같은 구조로 설계 할 수 있다.

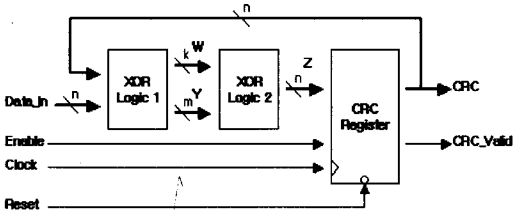


그림 3. 논리 레벨을 고려한 병렬 CRC 생성기 구조

3장에서 제시한 알고리즘의 3 단계로부터, 두 열에서 '1'을 삭제하면서 두 X를 입력으로 하고 Y를 출력으로 하는 하나의 게이트를 생성한다. X는 이미 D와 C를 입력으로 하는 2 입력 XOR 게이트의 출력이므로 D와 C에서 Y까지는 항상 두 논리 레벨을 유지하게 된다. 최종적으로 삭제되지 못하고 남은 '1'은 W를 통하여 다음 회로로 전송한다. 이렇게 하여 그림 3과 같이, 전체 XOR 로직을 X의 XOR 조합들로 이루어진 XOR Logic 1과, 중간 결과 W와 Y를 입력으로 하여, 최종적으로 레지스터에 저장 될 CRC 코드 Z를 생성하는 XOR Logic 2로 나눌 수 있다. XOR Logic 2 역시, 행은 Fi, 열은 Wk와 Ym의 연결(concatenation)인 새로운 행렬을 정의하고 다시 알고리즘을 적용함으로써 축소시킨다. 이렇게 함으로써, 2 입력 XOR 게이트를 사용하여 합성 할 경우, XOR Logic 1은 n의 크기에 상관없이 항상 두개의 논리 레벨로 구현되며, XOR Logic 2 또한 최소한의 논리 레벨로 구현이 가능하다. W와 Y의 데이터 폭 k와 m은 생성 다항식 (Polynomial), 알고리즘의 최적화 정도 및 하나의 XOR 게이트의 입력 수에 따라 달라질 것이다.

Ⅴ. 성능 평가 및 분석

성능 평가를 위하여 표 2와 같이, 널리 사용되고 있는 세가지 다항식을 이용하였으며, 어떤 알고리즘도 적용하지 않고 그대로 합성한 "Raw", 널리 사용되고 있는 로직 최적화 툴인 "SIS"를 이용한 합성 [9,10], 그리고 가장 최근 FPGA에서 뛰어난 성능을 보인 논문 "Ref.[8]"의 결과를 이 논문에서 제시한 알고리즘과 구조로 설계한 "HPC" (HPC:High Perfomance CRC)와 비교한다. Verilog HDL을 이용하여 코딩을 하고, Sinplicity의 "Synplify Pro 7.0"을 사용하여 FPGA 합성을 하였으며, Synopsys의 "Design Analyzer"를 사용하여 표준 셀 합성을 하였다. FPGA는, "Synplify Pro 7.0"가 Ref.[8]에서 사용한 LUT (Look-Up Table)방식의 Altera FPGA를 지원하지 않아서, LUT방식을 사용하는 FPGA중 Xilinx Virtex2 XC2V40을 선택했으며, 표준 셀 라이브러리는 Synopsys의 LSI_10K를 사용하였다. 표 3과 4가 합성 툴에 의한 성능 분석 결과이다.

표 2. Generating Polynomials

CRC-16 (ITU-TSS)	$P(X) = X^{16} + X^{12} + X^5 + 1$
CRC-16 (HDLC)	$P(X) = X^{16} + X^{15} + X^2 + 1$
CRC-32 (Ethernet)	$P(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X^1 + 1$

-. ITU-TSS : International Telecommunications Union-Telecommunications Standardization Sector

-. HDLC : High-level Data Link Control

"Ref.[8]"에 실어 놓은 VHDL 코드를 그대로 합성을 하였는데, Synopsys에서는 제대로 합성이 되지 않아서, 부득이 표 4에서 제외했다. 표 3에서 "SIS"와 "Ref.[8]"이 두개의 16 비트 CRC 회로에서는, "Raw"에 비하여 LUT수를 많이 줄여준 반면 속도가 오히려 크게 나왔으며, "HPC"는 모든 회로에서 LUT를 두드러지게 줄여주지는 못하지만 속도

표 3. Performance Comparisons I (with "Synplify Pro 7.0")

Polynomials		LUTs	Arrival Time (Critical Path)
CRC-16 (ITU-TSS)	Raw	38	3,861 ns
	SIS	38	4,688 ns
	Ref. [8]	31	4,774 ns
	HPC	37	3,703 ns
CRC-16 (HDLC)	Raw	46	4,403 ns
	SIS	33	4,532 ns
	Ref. [8]	24	4,562 ns
	HPC	32	3,290 ns
CRC-32 (Ethernet)	Raw	185	5,768 ns
	SIS	160	5,468 ns
	Ref. [8]	175	5,067 ns
	HPC	175	3,650 ns

-. LUT: Look Up Table

표 4. Performance Comparisons II (with "Synopsys Design Analyzer")

Polynomials		Total Cell Area	Arrival Time (Critical Path)
CRC-16 (ITU-TSS)	Raw	573	11.44 ns
	SIS	358	10.81 ns
	HPC	349	9.01 ns
CRC-16 (HDLC)	Raw	593	12.24 ns
	SIS	333	10.82 ns
	HPC	281	10.35 ns
CRC-32 (Ethernet)	Raw	3430	18.07 ns
	SIS	1451	14.36 ns
	HPC	1234	11.56 ns

면에 있어서는 매우 좋은 결과를 나타내고 있다. 표 4에서는 모든 회로에서 "SIS" 와 "HPC"가 "Raw"에 비해 셀 면적과 속도 면에 있어서 모두 월등히 나아짐을 보여주고 있다. "SIS"는 다중 레벨 로직 최적화 툴로써, 공유되지 않고 남은 X들이 W로 바이패스 되지 않고, Y와의 XOR로 또 하나의 Y 출력을 만들게 되어, 셀 수 및 논리 레벨이 증가하는 결과를 초래하기 때문에, 그러한 문제점을 해결한 "HPC"의 성능이 더 좋게 나온 것으로 분석된다.

VI. 결론 및 향후계획

논리 회로에서 게이트 수를 줄임으로써 최적화하는 방법은 NP-Hard 문제에 해당된다. 본 논문에서 제시한 알고리즘 또한 휴리스틱한 방법으로써 입출력을 나타내는 행렬로부터 보다 여러 출력에 영향을 끼치는 입력을 선택하여 좀더 공유될 가능성이 많은 게이트를 찾아 우선적으로 그룹화 하고자 하였다. 논리 레벨을 최소화하기 위한 구조로 설계함으로써 기존의 CRC 회로에 비해 처리속도가 매우 향상되었다. 그러나, 이 방법 역시CRC 와 같은 XOR 게이트로만 구성할 수 있는 특정회로에 적용한 휴리스틱 알고리즘 및 구현 방법이므로, 다른 최적화 알고리즘의 장단점을 분석하여 CRC 이외의 다양한 회로에 적용하여 성능을 향상시킬 수 있도록 알고리즘을 개선하고 일반화하는 연구를 계속할 것이다.

참 고 문 헌

[1] Cypress Semiconductor Corporation, "Parallel Cyclic Redundancy Check (CRC) for HOTLINKTM," Application note, Mar. 1999.

[2] R. N. Williams, "A Painless Guide to CRC Error Detection Algorithms," http://www.ross.net/crc, version 3, 19 Aug. 1993.

[3] T.-B. Pei and C. Zukowski, "High-Speed Parallel CRC Circuits in VLSI," *IEEE Trans. Commun.*, Vol. 40, no. 4, pp. 653-657, 1992.

[4] R. F. Hobson and K. L. Cheng, "A High-Performance CMOS 32-Bit Parallel CRC Engine," *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 2, pp. 233-235, Feb. 1999.

[5] S. M. Joshi, P. K. Dubey and M. A. Kaplan, "A New Parallel Algorithm for CRC Generation," *IEEE International Conference on Communications*, Vol. 3, pp. 18-22, Jun. 2000.

[6] M. D. Shieh et al., "A Systemic Approach for Parallel CRC Computations," *J. Information Science and Eng.*, May 2001.

[7] M. Spachmann, "Automatic Generation of

Parallel CRC Circuits," *IEEE Design and Test of Computers*, Vol. 18, pp. 108-114, May. 2001.

[8] G. Campobello, G. Patane and M. Russo, "Parallel CRC Realization," *IEEE Transactions on Computers*, Vol. 52, pp. 63-71, Oct. 2003.

[9] G. D. Micheli, *Sznthesis and Optimi ation of Digital Circuits.* McGRAW-HILL INTERNATIONAL EDITIONS, 1994.

[10] http://www-cad.eecs.berkeley.edu/Software/software.html

이 현 빈(Hyun-bean Yi) 정회원

2001년 2월 : 한양대학교 전자컴퓨터공학과 학사 졸업
2003년 2월 : 한양대학교 컴퓨터 공학과 석사 졸업
2003년 3월~현재 : 한양대학교 컴퓨터공학과 박사과정

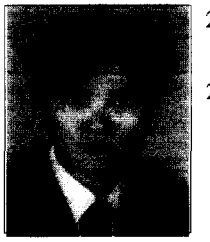
<관심분야> Digital System Design, Design For Testability, High Speed Interconnection Network

박 성 주(Sung-ju Park) 정회원

1988년 2월 : University of Massachusetts 전기컴퓨터공학 석사 졸업
1992년 2월 : University of Massachusetts 박사 졸업
1995년 3월~현재 : 한양대학교 전자컴퓨터공학과 부교수

<관심분야> 전자공학, 테스팅, 통신공학

민 병 우(Pyoung-woo Min) 정회원



2003년 2월 : 한양대학교 컴퓨터공학과 졸업
2003년 3월~현재 : 한양대학교 컴퓨터공학과 석사과정

<관심분야> Digital System Design, VLSI, 테스팅 High Speed Network

박 창 원(Chang-won Park)

1980년 3월 : 중앙대학교 전자공학과 졸업
2002년 8월 : 광운대학교 전자통신공학과 석사
1993년 7월~현재 : 전자부품연구원 수석연구원

<관심분야> 통신공학, 유비쿼터스